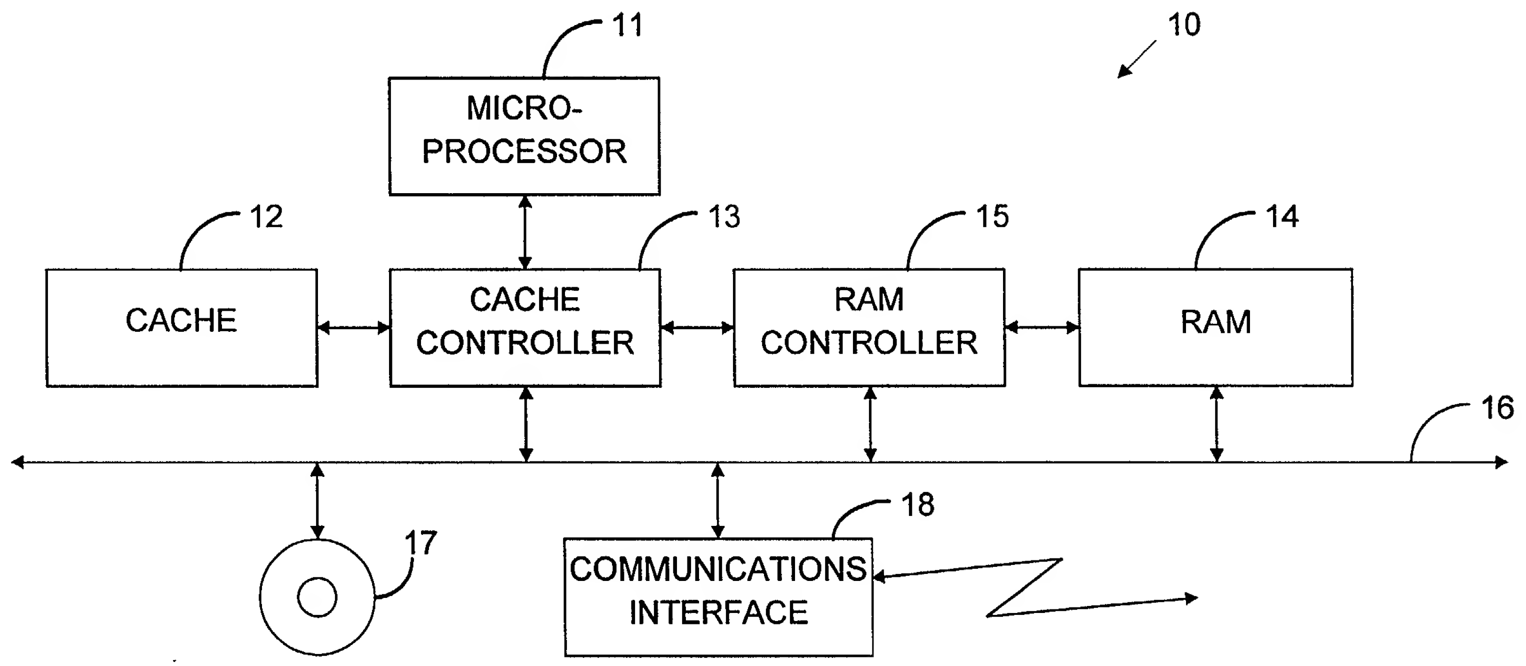

11
MICRO-PROCESSOR
10
12
CACHE
13
CACHE CONTROLLER
15
RAM CONTROLLER
14
RAM
16
17
18
COMMUNICATIONS INTERFACE

FIG. 1

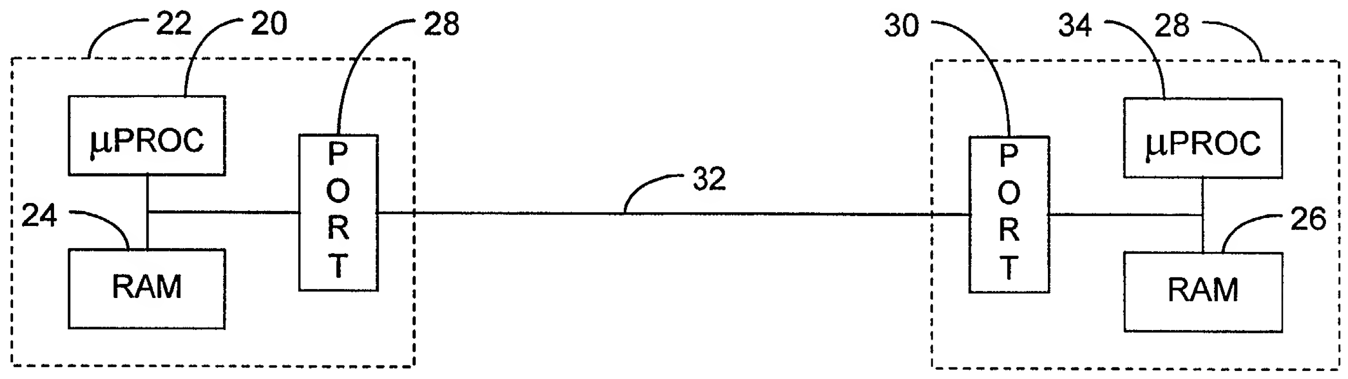

22
20
28
30
34
28
μPROC
24
RAM
PORT
32
PORT
μPROC
26
RAM

FIG. 2

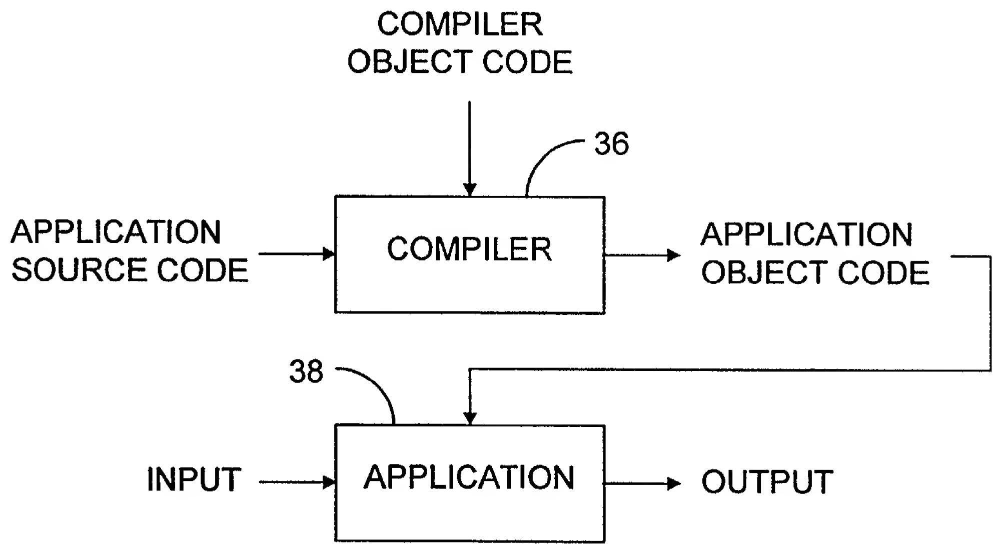

COMPILER OBJECT CODE
36
APPLICATION SOURCE CODE
COMPILER
APPLICATION OBJECT CODE
38
INPUT
APPLICATION
OUTPUT

FIG. 3

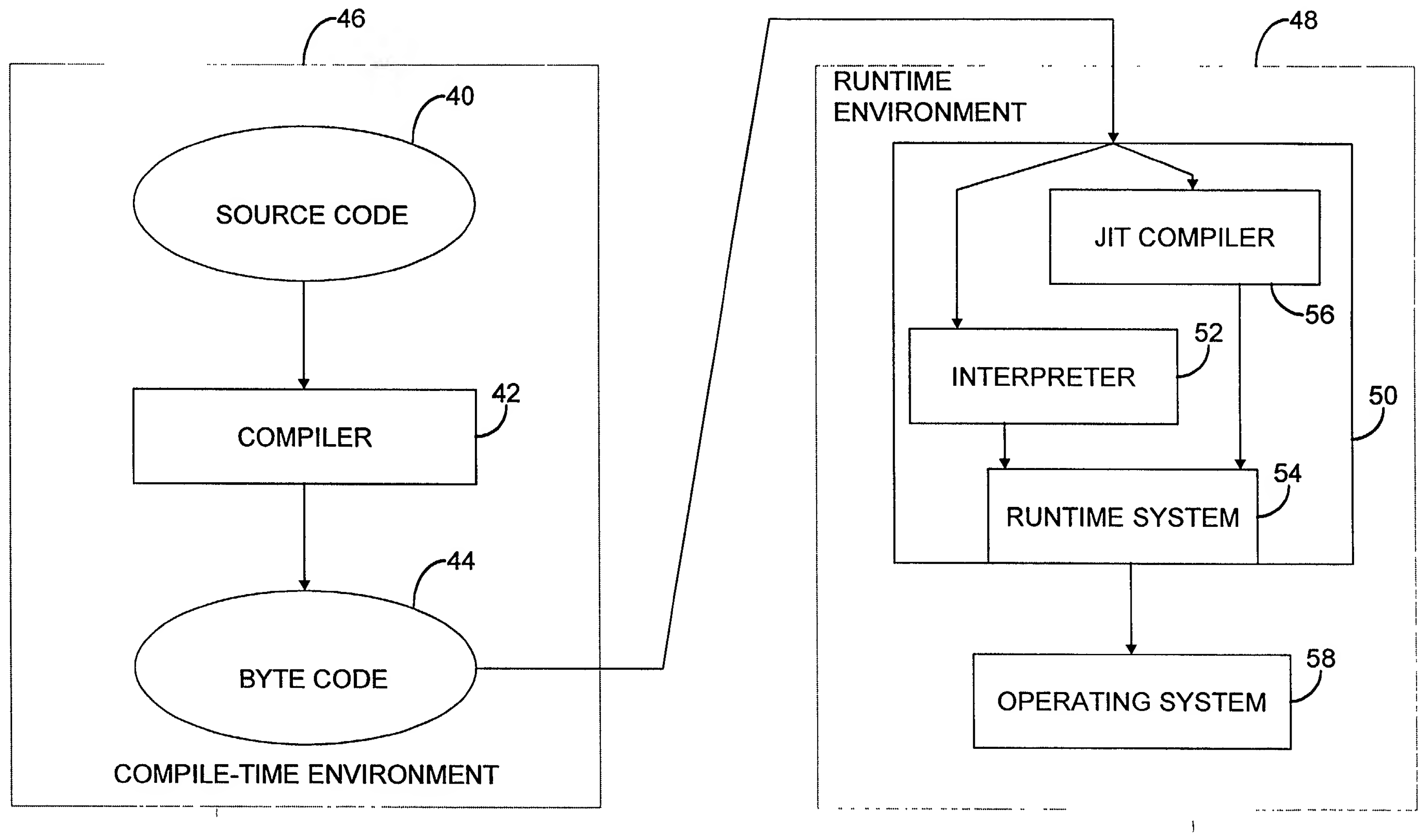
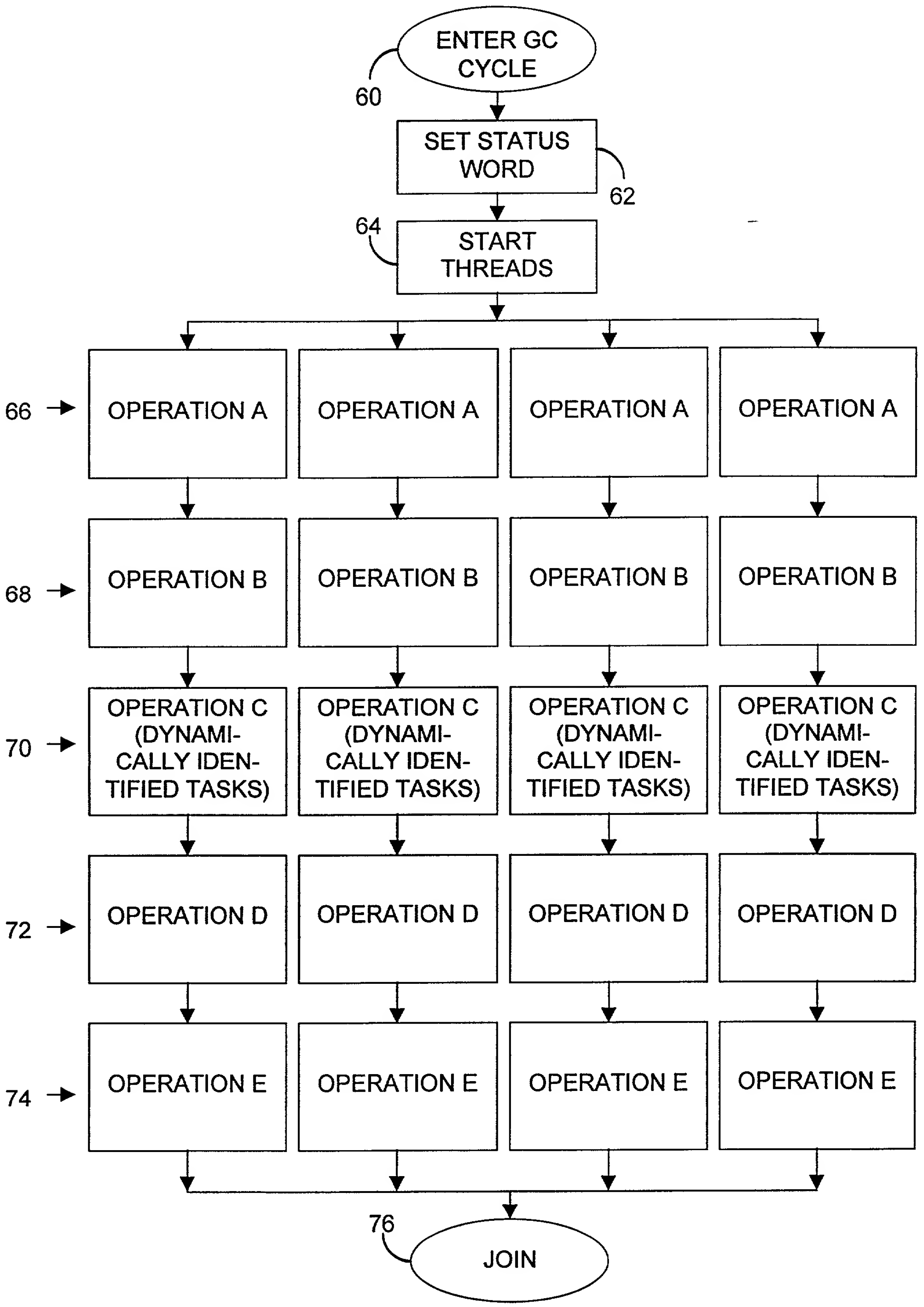
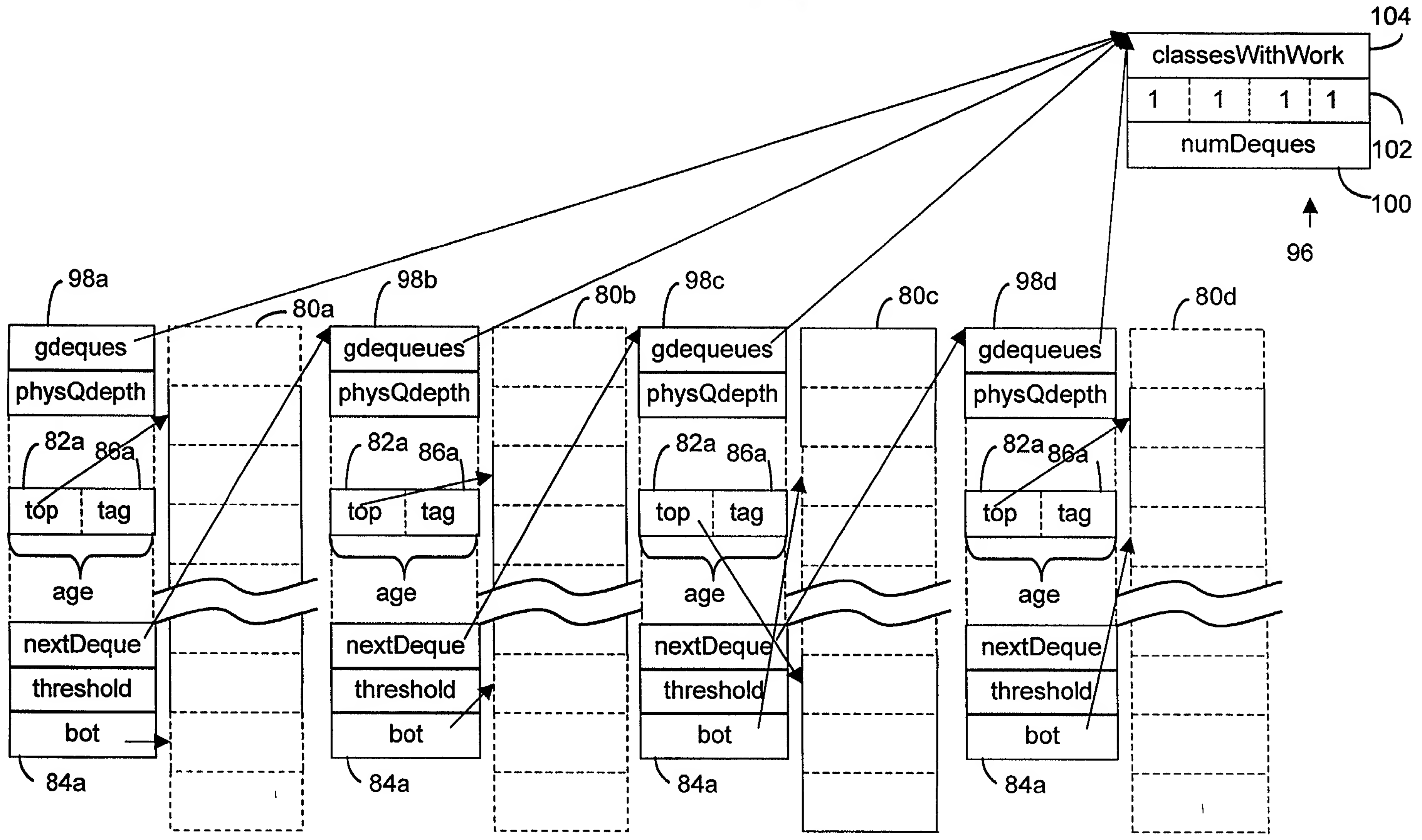

46
40
SOURCE CODE
COMPILER
42
44
BYTE CODE
COMPILE-TIME ENVIRONMENT
48
RUNTIME ENVIRONMENT
JIT COMPILER
56
INTERPRETER
52
50
RUNTIME SYSTEM
54
OPERATING SYSTEM
58

FIG. 4



ENTER GC CYCLE
60
SET STATUS WORD
62
64
START THREADS
66
OPERATION A
OPERATION A
OPERATION A
OPERATION A
68
OPERATION B
OPERATION B
OPERATION B
OPERATION B
70
OPERATION C (DYNAMI- CALLY IDEN- TIFIED TASKS)
OPERATION C (DYNAMI- CALLY IDEN- TIFIED TASKS)
OPERATION C (DYNAMI- CALLY IDEN- TIFIED TASKS)
OPERATION C (DYNAMI- CALLY IDEN- TIFIED TASKS)
72
OPERATION D
OPERATION D
OPERATION D
OPERATION D
74
OPERATION E
OPERATION E
OPERATION E
OPERATION E
76
JOIN

FIG. 5



classesWithWork
1
1
1
1
numDeques
104
102
100
96
98a
gdeques
physQdepth
82a
86a
top
tag
age
nextDeque
threshold
bot
84a
80a
98b
gdequeues
physQdepth
82a
86a
top
tag
age
nextDeque
threshold
bot
84a
80b
98c
gdequeues
physQdepth
82a
86a
top
tag
age
nextDeque
threshold
bot
84a
80c
98d
gdequeues
physQdepth
82a
86a
top
tag
age
nextDeque
threshold
bot
84a
80d

FIG. 6

```
static java_lang_Object * popTop (localDeque *dq){
        dequeAge oldAge = dq->age;
        unsigned int localBot = dq->bot;
        if (localBot == oldAge.top)
                return NULL;
        java_lang_Object * task = dq->objects[oldAge.top];
        dequeAge newAge = oldAge;
        if(++newAge.top==dq->physQdepth) newAge.top = 0;
        dequeueAge tempAge = (dequeAge) casInt((int) newAge,
                (int) oldAge,
                (int*) &dq->age); /*atomic compare-and-swap*/
        if(tempAge == oldAge)
                 return task;
        return NULL;
}
```

FIG. 7

```
static void dequePush (localDeque *dq, java_lang_Object *obj) {
    unsigned int localBot = dq->bot;
    dequeAge oldAge = dq->age;
    if (dequeNumberOfElements (localBot, oldAge.top, dq) == dq->physQdepth-1) {
        dequeOverflow(dq);
        localBot = dq->bot;
    }
    setArrayElement (localBot, dq, obj);
    if (++localBot == dq->physQdepth) localBot = 0;
    dq->bot = localBot;
}
```

```
static int dequeNumberOfElements(unsigned int localBot, unsigned int localTop,
            localDeque *dq) {
    int diff = localBot – localTop;
    if (diff < 0)
        diff = diff + dq->physQdepth;
        return diff;
}
```

```
static void setArrayElement(int index,
        localDeque *dq,
        java_lang_Object *obj) {
    dq->objects[index] = obj;
}
```

FIG. 8

```
static java_lang_Object *dequePopWork (localDeque *dq) {
    unsigned int localBot = dq->bot;
    java_lang_Object *obj = NULL;
    dequeAge oldAge, newAge;
    int numElems = dequeNumberOfElements (localBot, dq ->age.top, dq);
    if (numElems == 0)
        return NULL;
    if (numElems > dq->threshold) {
        while ((obj == NULL) &&
            (dequeNumberOfElements (localBot, dq->age.top, dq) > dq->threshold))
            {
                obj = popTop(dq)
            }
        if (obj != NULL)
            return obj;
    }
    if(--localBot == -1) localBot = dq->physQdepth - 1;
    dq->bot = localBot;
    obj = getArrayElement (localBot, dq);
    oldAge = dq->age; /* It might have changed */
    if (dequeNumberOfElements (localBot, oldAge.top, dq) > 0) return obj;
    newAge.tag = oldAge.tag + 1;
    newAge.top = oldAge.top
    if (localBot == oldAge.top) {
        dequeAge tempAge;
        tempAge = (dequeAge) casInt ((int) newAge,
             (int) oldAge,
            (int*) &dq->age);
    if (tempAge == oldAge) return obj;
    }
    dq->age = newAge;
    return NULL;
}
```

```
static void setArrayElement(int index,
        localDeque *dq,
        java_lang_Object *obj) {
    dq->objects[index] = obj;
}
```

FIG. 9

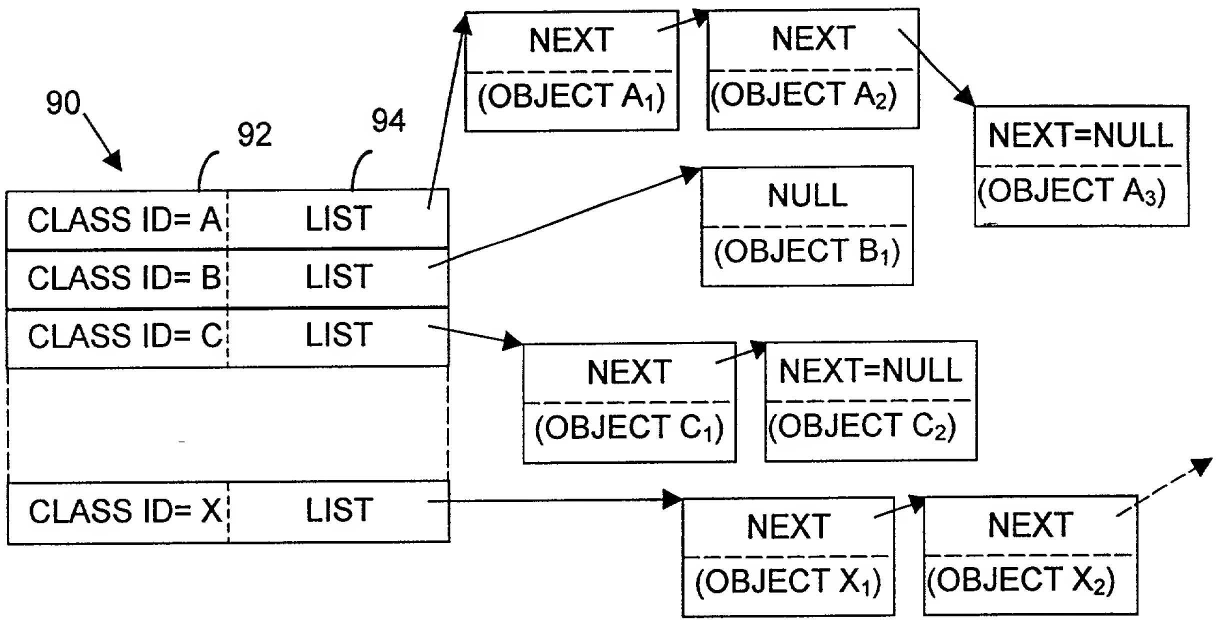

90
92
94
CLASS ID= A
LIST
CLASS ID= B
LIST
CLASS ID= C
LIST
CLASS ID= X
LIST
NEXT
(OBJECT A1)
NEXT
(OBJECT A2)
NEXT=NULL
(OBJECT A3)
NULL
(OBJECT B1)
NEXT
(OBJECT C1)
NEXT=NULL
(OBJECT C2)
NEXT
(OBJECT X1)
NEXT
(OBJECT X2)

FIG. 10

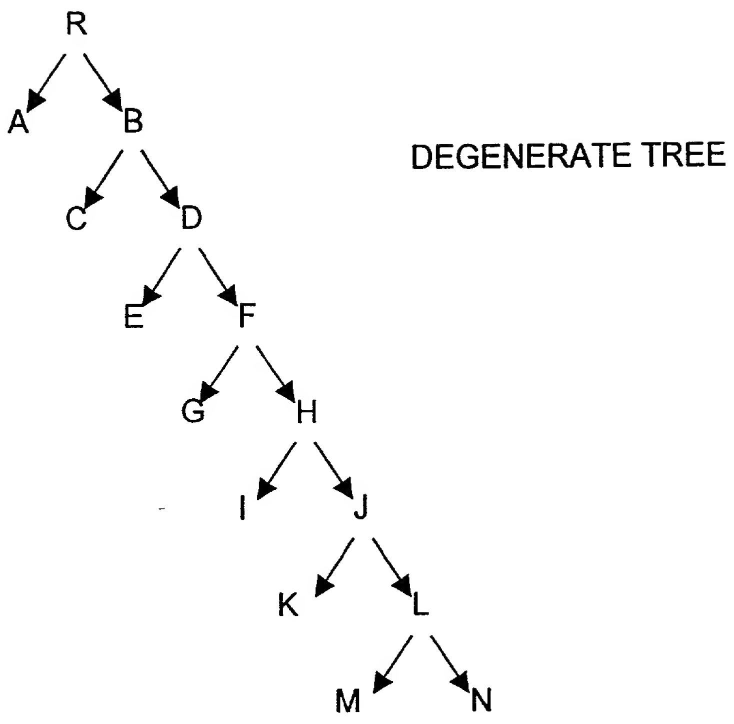


FIG. 11

LIFO SEQUENCE
R
AB
ACD
ACEF
ACEGH
ACEGIJ
ACEGIKL
ACEGIKMN
ACEGIKM
ACEGIK
ACEGI
ACEG
ACE
AC
A

FIG. 12

FIFO SEQUENCE
R
AB
B
CD
D
EF
F
GH
H
IJ
J
KL
L
MN
N

FIG. 13

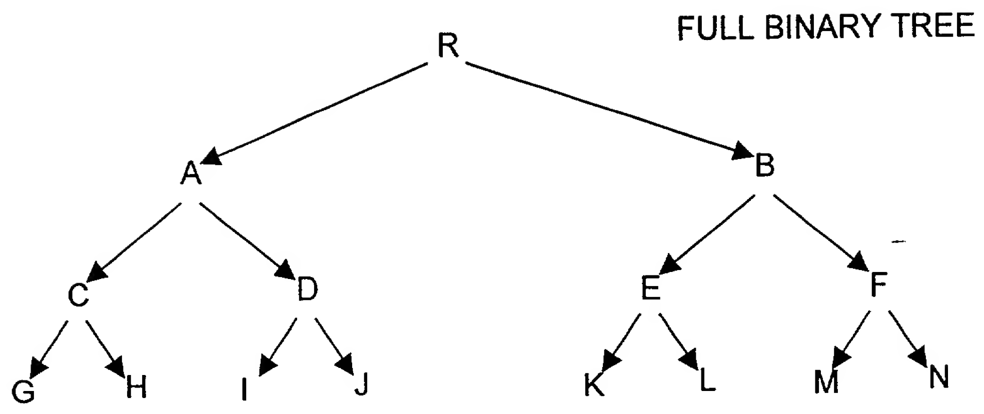


FIG.14

LIFO SEQUENCE
R
AB
AEF
AEMN
AEM
AE
AKL
AK
A
CD
CIJ
CI
C
GH
G

FIG. 15

FIFO SEQUENCE
R
AB
BCD
CDEF
DEFGH
EFGHIJ
FGHIJKL
GHIJKLMN
HIJKLMN
IJKLMN
JKLMN
KLMN
LMN
MN
N

FIG. 16

```
java_lang_Object *dequeFindWork(localDeque *dq) {
    java_lang_Object *result = findWorkHelper(dq);
    globalDeques *gdqs = dq->gdeques;
    if (result == NULL) {
        mark_self_inactive(dq->index, &gdqs->statusBitmap); /* You have no work */
    }
    while (result == NULL) {
        if (!gdqs->statusBitmap) return NULL;  /* No one has any work. Terminate. */
        poll(NULL, NULL, 0);
        if (checkForWork(dq)) { /* You don't have any work, but there is some either
                    on the overflow queue, or in another thread's work
                    queue */
            mark_self_active(dq->index, &gdqs->statusBitmap); /* Looking for work */
            result = findWorkHelper(dq);
            if (result == NULL) {
                mark_self_inactive(dq->index, &gdqs->statusBitmap);
            }
        }
    }
    return result;
}

java_lang_Object *findWorkHelper(localDeque *dq) {
    java_lang_Object *task = findWorkInOverflowList(dq);
    if (task == NULL) {
        task = stealWork(dq);
    }
    return task;
}

static void mark_self_inactive(int self, int *pStatusBitmap) {
        int oldValue,newValue;
        do {
                oldValue = *pStatusBitmap;
                newValue = oldValue & (~(1<<self));
                newValue = casInt(newValue, oldValue, pStatusBitmap);
        } while (newValue != oldValue) ;
}

static void mark_self_active(int self, int *pStatusBitmap) {
        int oldValue,newValue;
        do {
                oldValue = *pStatusBitmap;
                newValue = oldValue | (1<<self);
                newValue = casInt(newValue, oldValue, pStatusBitmap);
        } while (newValue != oldValue) ;
}
```

FIG. 17

```
static java_lang_Object *stealWork(localDeque *dq) {
    globalDeques *gdqs = dq->gdeques;
    int degree = gdqs->numDeques;
    int iterations = 2 * degree;
    int i = 0;
    while (i++ < iterations) {
        localDeque *dqToSteal = pickQueueToStealFrom(gdqs, dq);
        if (dqToSteal->bot > dqToSteal->age.top) {
            java_lang_Object *task = popTop(dqToSteal);
            if(!task) poll(NULL, NULL, 0);
            else return task;
        }
    }
    return NULL;
}
```

FIG. 18

```
static bool_t checkForWork(localDeque *dq) {
    globalDeques *gdqs = dq->gdeques;
    return gdqs->classesWithWork || peekDeque(dq);
}
```

```
static bool_t peekDeque(localDeque *dq) {
   globalDeques *gdqs = dq->gdeques;
   int degree = gdqs->numDeques;
   int i;
   for (i = 0; i < 2 * degree; i++) {
      localDeque *dqToPeek = pickQueueToStealFrom(gdqs, dq);
      if (dqToPeek->bot > dqToPeek->age.top) {
        return TRUE;
      }
   }
   return FALSE;
}
```

FIG. 19